Technical Disclosure Commons

Defensive Publications Series

January 2021

Locating and Securing Use-After-Free Pointers

Kentaro Hara

Follow this and additional works at: https://www.tdcommons.org/dpubs_series

Recommended Citation
Hara, Kentaro, "Locating and Securing Use-After-Free Pointers", Technical Disclosure Commons, (January 06, 2021)
https://www.tdcommons.org/dpubs_series/3943

This work is licensed under a Creative Commons Attribution 4.0 License.
This Article is brought to you for free and open access by Technical Disclosure Commons. It has been accepted for inclusion in Defensive Publications Series by an authorized administrator of Technical Disclosure Commons.

Locating and Securing Use-After-Free Pointers

ABSTRACT

Dangling pointers and/or use-after-free pointers are gateways to exploits for software such as browsers, operating systems, etc. To forestall attacks that rely on dangling or user-after-free pointers, ordinary pointers in source code can be replaced by secure pointers. However, rolling out code with such replacement requires the developer to incur testing and engineering costs. Further, due to the likely memory and runtime overheads of secure pointers, it is useful to identify ordinary pointers in source code that merit replacement. This disclosure describes techniques to determine which ordinary pointers in source code are suitable for replacement by secure pointers.

KEYWORDS

- Dangling pointer
- Secure pointer
- Use-after-free memory
- Use-after-free attack
- Memory management
- Memory leak
- Program crash
- Crash report

BACKGROUND

When the memory for an object is deallocated, pointers to that object become invalid. Such pointers, known as dangling pointers, can be exploited by attackers to cause undesirable or malicious program behavior. A pointer that points to memory that has been reused (reallocated) is known as a use-after-free pointer; these, too, are gateways to exploits.

A previous publication [1] described techniques to track pointers at runtime and periodically test pointers to determine if they are pointing to deallocated or reallocated memory. Another publication [2] described techniques to detect the presence of dangling and use-after-

free pointers by augmenting pointer and pointed-to object with metadata that enables the checking of the validity of the pointer and the reuse status of the memory it points to. In both techniques, upon the discovery of dangling or use-after-free pointers, security breaches are forestalled by causing a program crash, which may be a preferred option to leaving open the potential for security breaches.

In both techniques, ordinary pointers are replaced by secure pointer classes that are wrappers around ordinary pointers. For example, in [1], a pointer class `SafePtr` is defined that encapsulates the ordinary pointer and an additional pointer to a linked-list that serves as a pointer-tracking mechanism. In [2], a pointer class `CheckedPtr` is defined that encapsulates the ordinary pointer and additional metadata to enable the checking of the validity of the pointer and the reuse status of the memory it points to. In both techniques, ordinary pointers in the source code are replaced selectively or globally by the new secure pointer classes.

In complex software, e.g., browsers, operating systems, etc., with thousands of modules and perhaps tens of thousands of pointer variables, replacement of an ordinary pointer by a `SafePtr` or a `CheckedPtr` would invariably involve testing and engineering costs. Although the memory and runtime overheads and engineering costs of `SafePtr` and `CheckedPtr` are likely modest, it is still useful to reduce overhead as much as possible. Not all ordinary pointers in source code merit replacement. Optimally, only those ordinary pointers that are likely to become dangling or use-after-free (and hence, sources of potential security breaches) need to be replaced by the more secure `SafePtr` and `CheckedPtr` classes. However, it is difficult to know which ordinary pointer stands as a gateway to a potential security breach.

DESCRIPTION

This disclosure describes techniques to determine the ordinary pointers in source code that are to be replaced by more secure pointer classes.

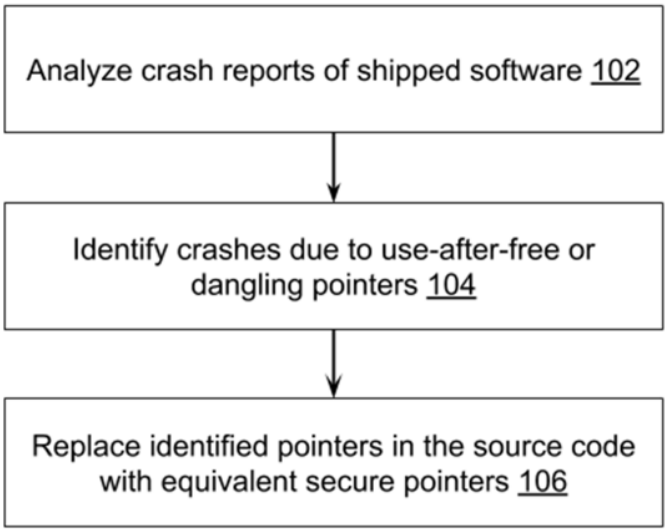

Fig. 1: Locating and securing use-after-free and dangling pointers

Fig. 1 illustrates locating and securing use-after-free and dangling pointers in production (shipped) code, per the techniques of this disclosure. During normal usage of shipped software, the occasional bug can cause a crash, which is usually accompanied by a crash report. Users are provided with options to send such crash reports to the developer, who can analyze (102) the reports to identify crashes that are caused by use-after-free or dangling pointers (104). For example, crash reports can include data that enables such identification. Pointers that are determined to have been the cause for crashes are replaced in the source code with equivalent secure pointers (106), e.g., `SafePtr` or `CheckedPtr` in a subsequent version.

CONCLUSION

Dangling pointers and/or use-after-free pointers are gateways to exploits for software such as browsers, operating systems, etc. To forestall attacks that rely on dangling or user-after-free pointers, ordinary pointers in source code can be replaced by secure pointers. However, rolling out code with such replacement requires the developer to incur testing and engineering costs. Further, due to the likely memory and runtime overheads of secure pointers are modest, it is useful to identify ordinary pointers in source code that merit replacement. This disclosure describes techniques to determine which ordinary pointers in source code are suitable for replacement by secure pointers.

REFERENCES

1. Hara, Kentaro, "Automatic Sanitization of Use-After-Free Pointers", Technical Disclosure Commons, (January 04, 2021) https://www.tdcommons.org/dpubs_series/3932
2. Hara, Kentaro and Lizé, Benoît, "Detecting Dangling Pointers Using Embedded Metadata", Technical Disclosure Commons, (December 30, 2020) https://www.tdcommons.org/dpubs_series/3931